

# Study guide: Computing with variational forms

Hans Petter Langtangen<sup>1,2</sup>

<sup>1</sup>Center for Biomedical Computing, Simula Research Laboratory

<sup>2</sup>Department of Informatics, University of Oslo

Jun 7, 2015

## Basic principles for approximating differential equations

We shall apply least squares, Galerkin/projection, and collocation to differential equation models

Our aim is to extend the ideas for approximating  $f$  by  $u$ , or solving

$$u = f$$

to real, *spatial* differential equations like

$$-u'' + bu = f, \quad u(0) = C, \quad u'(L) = D$$

Emphasis will be on the Galerkin/projection method
--

## Abstract differential equation

$$\mathcal{L}(u) = 0, \quad x \in \Omega$$

Examples (1D problems):

$$\mathcal{L}(u) = \frac{d^2 u}{dx^2} - f(x),$$

$$\mathcal{L}(u) = \frac{d}{dx} \left( \alpha(x) \frac{du}{dx} \right) + f(x),$$

$$\mathcal{L}(u) = \frac{d}{dx} \left( \alpha(u) \frac{du}{dx} \right) - au + f(x),$$

$$\mathcal{L}(u) = \frac{d}{dx} \left( \alpha(u) \frac{du}{dx} \right) + f(u, x)$$

## Abstract boundary conditions

$$\mathcal{B}_0(u) = 0, \quad x = 0, \quad \mathcal{B}_1(u) = 0, \quad x = L$$

Examples:

$$\mathcal{B}_i(u) = u - g, \quad \text{Dirichlet condition}$$

$$\mathcal{B}_i(u) = -\alpha \frac{du}{dx} - g, \quad \text{Neumann condition}$$

$$\mathcal{B}_i(u) = -\alpha \frac{du}{dx} - h(u - g), \quad \text{Robin condition}$$

## Reminder about notation

- $u_e(x)$  is the symbol for the *exact* solution of  $\mathcal{L}(u_e) = 0 + \mathcal{B}_i = 0$
- $u(x)$  denotes an *approximate* solution
- $V = \text{span}\{\psi_0(x), \dots, \psi_N(x)\}$ ,  $V$  has basis  $\{\psi_i\}_{i \in \mathcal{I}_s}$
- We seek  $u \in V$
- $\mathcal{I}_s = \{0, \dots, N\}$  is an index set
- $u(x) = \sum_{j \in \mathcal{I}_s} c_j \psi_j(x)$
- Inner product:  $(u, v) = \int_{\Omega} uv \, dx$
- Norm:  $\|u\| = \sqrt{(u, u)}$

## New topics: variational formulation and boundary conditions

Much is similar to approximating a function (solving  $u = f$ ), but two new topics are needed:

- Variational formulation of the differential equation problem (including integration by parts)
- Handling of boundary conditions

## Residual-minimizing principles

- When solving  $u = f$  we knew the error  $e = f - u$  and could use principles for minimizing the error
- When solving  $\mathcal{L}(u_e) = 0$  we do not know  $u_e$  and cannot work with the error  $e = u_e - u$
- We can only know the *error in the equation*: the residual  $R$

Inserting  $u = \sum_j c_j \psi_j$  in  $\mathcal{L} = 0$  gives a residual  $R$

$$\mathcal{L}(u) = \mathcal{L}\left(\sum_j c_j \psi_j\right) = R \neq 0$$

Goal: minimize  $R$  with respect to  $\{c_i\}_{i \in \mathcal{I}_s}$  (and hope it makes a small  $e$  too)

$$R = R(c_0, \dots, c_N; x)$$

## The least squares method

Idea: minimize

$$E = \|R\|^2 = (R, R) = \int_{\Omega} R^2 dx$$

Minimization wrt  $\{c_i\}_{i \in \mathcal{I}_s}$  implies

$$\frac{\partial E}{\partial c_i} = \int_{\Omega} 2R \frac{\partial R}{\partial c_i} dx = 0 \quad \Leftrightarrow \quad \left(R, \frac{\partial R}{\partial c_i}\right) = 0, \quad i \in \mathcal{I}_s$$

$N + 1$  equations for  $N + 1$  unknowns  $\{c_i\}_{i \in \mathcal{I}_s}$

## The Galerkin method

Idea: make  $R$  orthogonal to  $V$ ,

$$(R, v) = 0, \quad \forall v \in V$$

This implies

$$(R, \psi_i) = 0, \quad i \in \mathcal{I}_s$$

$N + 1$  equations for  $N + 1$  unknowns  $\{c_i\}_{i \in \mathcal{I}_s}$

## The Method of Weighted Residuals

Generalization of the Galerkin method: demand  $R$  orthogonal to some space  $W$ , possibly  $W \neq V$ :

$$(R, v) = 0, \quad \forall v \in W$$

If  $\{w_0, \dots, w_N\}$  is a basis for  $W$ :

$$(R, w_i) = 0, \quad i \in \mathcal{I}_s$$

- $N + 1$  equations for  $N + 1$  unknowns  $\{c_i\}_{i \in \mathcal{I}_s}$
- Weighted residual with  $w_i = \partial R / \partial c_i$  gives least squares

## New terminology: test and trial functions

- $\psi_j$  used in  $\sum_j c_j \psi_j$  is called *trial function*
- $\psi_i$  or  $w_i$  used as weight in Galerkin's method is called *test function*

## The collocation method

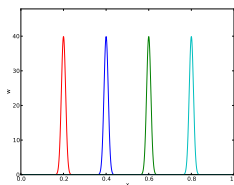
Idea: demand  $R = 0$  at  $N + 1$  points in space

$$R(x_i; c_0, \dots, c_N) = 0, \quad i \in \mathcal{I}_s$$

The collocation method is a weighted residual method with delta functions as weights

$$0 = \int_{\Omega} R(x; c_0, \dots, c_N) \delta(x - x_i) dx = R(x_i; c_0, \dots, c_N)$$

$$\text{property of } \delta(x) : \quad \int_{\Omega} f(x) \delta(x - x_i) dx = f(x_i), \quad x_i \in \Omega$$



## Examples on using the principles

**Goal.** Exemplify the least squares, Galerkin, and collocation methods in a simple 1D problem with global basis functions.

### The first model problem

$$-u''(x) = f(x), \quad x \in \Omega = [0, L], \quad u(0) = 0, \quad u(L) = 0$$

Basis functions:

$$\psi_i(x) = \sin\left((i+1)\pi\frac{x}{L}\right), \quad i \in \mathcal{I}_s$$

Residual:

$$\begin{aligned} R(x; c_0, \dots, c_N) &= u''(x) + f(x), \\ &= \frac{d^2}{dx^2} \left( \sum_{j \in \mathcal{I}_s} c_j \psi_j(x) \right) + f(x), \\ &= - \sum_{j \in \mathcal{I}_s} c_j \psi_j''(x) + f(x) \end{aligned}$$

### Boundary conditions

Since  $u(0) = u(L) = 0$  we must ensure that all  $\psi_i(0) = \psi_i(L) = 0$ , because then

$$u(0) = \sum_j c_j \psi_j(\mathbf{0}) = 0, \quad u(L) = \sum_j c_j \psi_j(L) = 0$$

- $u$  known: Dirichlet boundary condition
- $u'$  known: Neumann boundary condition
- Must have  $\psi_i = 0$  where Dirichlet conditions apply

### The least squares method; principle

$$\left(R, \frac{\partial R}{\partial c_i}\right) = 0, \quad i \in \mathcal{I}_s$$

$$\frac{\partial R}{\partial c_i} = \frac{\partial}{\partial c_i} \left( \sum_{j \in \mathcal{I}_s} c_j \psi_j''(x) + f(x) \right) = \psi_i''(x)$$

Because:

$$\frac{\partial}{\partial c_i} (c_0 \psi_0'' + c_1 \psi_1'' + \dots + c_{i-1} \psi_{i-1}'' + c_i \psi_i'' + c_{i+1} \psi_{i+1}'' + \dots + c_N \psi_N'') = \psi_i''$$

### The least squares method; equation system

$$\left(\sum_j c_j \psi_j'' + f, \psi_i''\right) = 0, \quad i \in \mathcal{I}_s$$

Rearrangement:

$$\sum_{j \in \mathcal{I}_s} (\psi_i'', \psi_j'') c_j = -(f, \psi_i''), \quad i \in \mathcal{I}_s$$

This is a linear system

$$\sum_{j \in \mathcal{I}_s} A_{i,j} c_j = b_i, \quad i \in \mathcal{I}_s$$

### The least squares method; matrix and right-hand side expressions

$$\begin{aligned} A_{i,j} &= (\psi_i'', \psi_j'') \\ &= \pi^4 (i+1)^2 (j+1)^2 L^{-4} \int_0^L \sin\left((i+1)\pi \frac{x}{L}\right) \sin\left((j+1)\pi \frac{x}{L}\right) dx \\ &= \begin{cases} \frac{1}{2} L^{-3} \pi^4 (i+1)^4 & i = j \\ 0, & i \neq j \end{cases} \\ b_i &= -(f, \psi_i'') = (i+1)^2 \pi^2 L^{-2} \int_0^L f(x) \sin\left((i+1)\pi \frac{x}{L}\right) dx \end{aligned}$$

### Orthogonality of the basis functions gives diagonal matrix

Useful property of the chosen basis functions:

$$\int_0^L \sin\left((i+1)\pi \frac{x}{L}\right) \sin\left((j+1)\pi \frac{x}{L}\right) dx = \delta_{ij}, \quad \delta_{ij} = \begin{cases} \frac{1}{2} L & i = j \\ 0, & i \neq j \end{cases}$$

$\Rightarrow (\psi_i'', \psi_j'') = \delta_{ij}$ , i.e., diagonal  $A_{i,j}$ , and we can easily solve for  $c_i$ :

$$c_i = \frac{2L}{\pi^2 (i+1)^2} \int_0^L f(x) \sin\left((i+1)\pi \frac{x}{L}\right) dx$$

### Least squares method; solution

Let sympy do the work ( $f(x) = 2$ ):

```
from sympy import *
import sys

i, j = symbols('i j', integer=True)
```

```

x, L = symbols('x L')
f = 2
a = 2*L/(pi**2*(i+1)**2)
c_i = a*integrate(f*sin((i+1)*pi*x/L), (x, 0, L))
c_i = simplify(c_i)
print c_i

```

$$c_i = 4 \frac{L^2 \left( (-1)^i + 1 \right)}{\pi^3 (i^3 + 3i^2 + 3i + 1)}, \quad u(x) = \sum_{k=0}^{N/2} \frac{8L^2}{\pi^3 (2k+1)^3} \sin \left( (2k+1) \pi \frac{x}{L} \right)$$

Fast decay:  $c_2 = c_0/27$ ,  $c_4 = c_0/125$  - only one term might be good enough:

$$u(x) \approx \frac{8L^2}{\pi^3} \sin \left( \pi \frac{x}{L} \right)$$

## The Galerkin method; principle

$R = u'' + f$ :

$$(u'' + f, v) = 0, \quad \forall v \in V,$$

or rearranged,

$$(u'', v) = -(f, v), \quad \forall v \in V$$

This is a *variational formulation* of the differential equation problem.

$\forall v \in V$  is equivalent with  $\forall v \in \psi_i$ ,  $i \in \mathcal{I}_s$ , resulting in

$$\left( \sum_{j \in \mathcal{I}_s} c_j \psi_j'', \psi_i \right) = -(f, \psi_i), \quad i \in \mathcal{I}_s$$

$$\sum_{j \in \mathcal{I}_s} (\psi_j'', \psi_i) c_j = -(f, \psi_i), \quad i \in \mathcal{I}_s$$

## The Galerkin method; solution

Since  $\psi_i'' \propto -\psi_i$ , Galerkin's method gives the same linear system and the same solution as the least squares method (in this particular example).

## The collocation method

$R = 0$  (i.e., the differential equation) must be satisfied at  $N + 1$  points:

$$-\sum_{j \in \mathcal{I}_s} c_j \psi_j''(x_i) = f(x_i), \quad i \in \mathcal{I}_s$$

This is a linear system  $\sum_j A_{i,j} = b_i$  with entries

$$A_{i,j} = -\psi_j''(x_i) = (j+1)^2 \pi^2 L^{-2} \sin\left((j+1)\pi \frac{x_i}{L}\right), \quad b_i = 2$$

Choose:  $N = 0$ ,  $x_0 = L/2$

$$c_0 = 2L^2/\pi^2$$

## Comparison of the methods

- Exact solution:  $u(x) = x(L-x)$
- Galerkin or least squares ( $N = 0$ ):  $u(x) = 8L^2\pi^{-3} \sin(\pi x/L)$
- Collocation method ( $N = 0$ ):  $u(x) = 2L^2\pi^{-2} \sin(\pi x/L)$ .

```
>> import sympy as sym
>> # Computing with Dirichlet conditions: -u''=2 and sines
>> x, L = sym.symbols('x L')
>> e_Galerkin = x*(L-x) - 8*L**2*sym.pi**(-3)*sym.sin(sym.pi*x/L)
>> e_colloc = x*(L-x) - 2*L**2*sym.pi**(-2)*sym.sin(sym.pi*x/L)

>> # Verify max error for x=L/2
>> dedx_Galerkin = sym.diff(e_Galerkin, x)
>> dedx_Galerkin.subs(x, L/2)
0
>> dedx_colloc = sym.diff(e_colloc, x)
>> dedx_colloc.subs(x, L/2)
0

# Compute max error: x=L/2, evaluate numerical, and simplify
>> sym.simplify(e_Galerkin.subs(x, L/2).evalf(n=3))
-0.00812*L**2
>> sym.simplify(e_colloc.subs(x, L/2).evalf(n=3))
0.0473*L**2
```

## Useful techniques

### Integration by parts has many advantages

Second-order derivatives will hereafter be integrated by parts

$$\begin{aligned} \int_0^L u''(x)v(x)dx &= - \int_0^L u'(x)v'(x)dx + [vu']_0^L \\ &= - \int_0^L u'(x)v'(x)dx + u'(L)v(L) - u'(0)v(0) \end{aligned}$$

Motivation:

- Lowers the order of derivatives



- Gives more symmetric forms (incl. matrices)
- Enables easy handling of Neumann boundary conditions
- Finite element basis functions  $\varphi_i$  have discontinuous derivatives (at cell boundaries) and are not suited for terms with  $\varphi_i''$

### We use a boundary function to deal with non-zero Dirichlet boundary conditions

- What about nonzero Dirichlet conditions? Say  $u(L) = D$
- We always require  $\psi_i(L) = 0$  (i.e.,  $\psi_i = 0$  where Dirichlet conditions applies)
- Problem:  $u(L) = \sum_j c_j \psi_j(L) = \sum_j c_j \cdot 0 = 0 \neq D$  - always!
- Solution:  $u(x) = B(x) + \sum_j c_j \psi_j(x)$
- $B(x)$ : user-constructed boundary function that fulfills the Dirichlet conditions
- If  $u(L) = D$ , make sure  $B(L) = D$
- No restrictions of how  $B(x)$  varies in the interior of  $\Omega$

### Example on constructing a boundary function for two Dirichlet conditions

Dirichlet conditions:  $u(0) = C$  and  $u(L) = D$ . Choose for example

$$B(x) = \frac{1}{L}(C(L-x) + Dx) : \quad B(0) = C, \quad B(L) = D$$

$$u(x) = B(x) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x),$$

$$u(0) = B(0) = C, \quad u(L) = B(L) = D$$

### Example on constructing a boundary function for one Dirichlet conditions

Dirichlet condition:  $u(L) = D$ . Choose for example

$$B(x) = D : \quad B(L) = D$$

$$u(x) = B(x) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x),$$

$$u(L) = B(L) = D$$

**With a  $B(x)$ ,  $u \notin V$ , but  $\sum_j c_j \psi_j \in V$**

- $\{\psi_i\}_{i \in \mathcal{I}_s}$  is a basis for  $V$
- $\sum_{j \in \mathcal{I}_s} c_j \psi_j(x) \in V$
- But  $u \notin V$ !
- Reason: say  $u(0) = C$  and  $u \in V$ ; any  $v \in V$  has  $v(0) = C$ , then  $2u \notin V$  because  $2u(0) = 2C$  (wrong value)
- When  $u(x) = B(x) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x)$ ,  $B \notin V$  (in general) and  $u \notin V$ , but  $(u - B) \in V$  since  $\sum_j c_j \psi_j \in V$

### Abstract notation for variational formulations

The finite element literature (and much FEniCS documentation) applies an abstract notation for the variational formulation:

Find  $(u - B) \in V$  such that

$$a(u, v) = L(v) \quad \forall v \in V$$

### Example on abstract notation

$$-u'' = f, \quad u'(0) = C, \quad u(L) = D, \quad u = D + \sum_j c_j \psi_j$$

Variational formulation:

$$\int_{\Omega} u' v' dx = \int_{\Omega} f v dx - v(0)C \quad \text{or} \quad (u', v') = (f, v) - v(0)C \quad \forall v \in V$$

Abstract formulation: find  $(u - B) \in V$  such that

$$a(u, v) = L(v) \quad \forall v \in V$$

We identify

$$a(u, v) = (u', v'), \quad L(v) = (f, v) - v(0)C$$

## Bilinear and linear forms

- $a(u, v)$  is a *bilinear form*
- $L(v)$  is a *linear form*

Linear form means

$$L(\alpha_1 v_1 + \alpha_2 v_2) = \alpha_1 L(v_1) + \alpha_2 L(v_2),$$

Bilinear form means

$$\begin{aligned} a(\alpha_1 u_1 + \alpha_2 u_2, v) &= \alpha_1 a(u_1, v) + \alpha_2 a(u_2, v), \\ a(u, \alpha_1 v_1 + \alpha_2 v_2) &= \alpha_1 a(u, v_1) + \alpha_2 a(u, v_2) \end{aligned}$$

In nonlinear problems: Find  $(u - B) \in V$  such that  $F(u; v) = 0 \quad \forall v \in V$

## The linear system associated with the abstract form

$$a(u, v) = L(v) \quad \forall v \in V \quad \Leftrightarrow \quad a(u, \psi_i) = L(\psi_i) \quad i \in \mathcal{I}_s$$

We can now derive the corresponding linear system once and for all by inserting  $u = B + \sum_j c_j \psi_j$ :

$$a(B + \sum_{j \in \mathcal{I}_s} c_j \psi_j, \psi_i) c_j = L(\psi_i) \quad i \in \mathcal{I}_s$$

Because of linearity,

$$\sum_{j \in \mathcal{I}_s} \underbrace{a(\psi_j, \psi_i)}_{A_{i,j}} c_j = \underbrace{L(\psi_i) - a(B, \psi_i)}_{b_i} \quad i \in \mathcal{I}_s$$

## Equivalence with minimization problem

If  $a$  is symmetric:  $a(u, v) = a(v, u)$ ,

$$a(u, v) = L(v) \quad \forall v \in V$$

is equivalent to minimizing the functional

$$F(v) = \frac{1}{2} a(v, v) - L(v)$$

over all functions  $v \in V$ . That is,

$$F(u) \leq F(v) \quad \forall v \in V$$

- Much used in the early days of finite elements
- Still much used in structural analysis and elasticity
- Not as general as Galerkin's method (since we require  $a(u, v) = a(v, u)$ )

## Examples on variational formulations

**Goal.** Derive variational formulations for some prototype differential equations in 1D that include

- variable coefficients
- mixed Dirichlet and Neumann conditions
- nonlinear coefficients

### Variable coefficient; problem

$$-\frac{d}{dx} \left( \alpha(x) \frac{du}{dx} \right) = f(x), \quad x \in \Omega = [0, L], \quad u(0) = C, \quad u(L) = D$$

- Variable coefficient  $\alpha(x)$
- $V = \text{span}\{\psi_0, \dots, \psi_N\}$
- *Nonzero* Dirichlet conditions at  $x = 0$  and  $x = L$
- Must have  $\psi_i(0) = \psi_i(L) = 0$
- Any  $v \in V$  has then  $v(0) = v(L) = 0$
- $B(x) = C + \frac{1}{L}(D - C)x$

$$u(x) = B(x) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x),$$

### Variable coefficient; Galerkin principle

$$R = -\frac{d}{dx} \left( \alpha \frac{du}{dx} \right) - f$$

Galerkin's method:

$$(R, v) = 0, \quad \forall v \in V$$

or with integrals:

$$\int_{\Omega} \left( -\frac{d}{dx} \left( \alpha \frac{du}{dx} \right) - f \right) v \, dx = 0, \quad \forall v \in V$$

### Variable coefficient; integration by parts

$$-\int_{\Omega} \frac{d}{dx} \left( \alpha(x) \frac{du}{dx} \right) v \, dx = \int_{\Omega} \alpha(x) \frac{du}{dx} \frac{dv}{dx} \, dx - \left[ \alpha \frac{du}{dx} v \right]_0^L$$

Boundary terms vanish since  $v(0) = v(L) = 0$

### Variable coefficient; variational formulation

**Variational formulation.** Find  $(u - B) \in V$  such that

$$\int_{\Omega} \alpha(x) \frac{du}{dx} \frac{dv}{dx} dx = \int_{\Omega} f(x) v dx, \quad \forall v \in V$$

Compact notation:

$$\underbrace{(\alpha u', v')}_{a(u, v)} = \underbrace{(f, v)}_{L(v)}, \quad \forall v \in V$$

### Variable coefficient; linear system (the easy way)

With

$$a(u, v) = (\alpha u', v'), \quad L(v) = (f, v)$$

we can just use the formula for the linear system:

$$A_{i,j} = a(\psi_j, \psi_i) = (\alpha \psi_j', \psi_i') = \int_{\Omega} \alpha \psi_j' \psi_i' dx = \int_{\Omega} \psi_i' \alpha \psi_j' dx \quad (= a(\psi_i, \psi_j) = A_{j,i})$$

$$b_i = (f, \psi_i) - (\alpha B', \psi_i) = \int_{\Omega} (f \psi_i - \alpha L^{-1}(D - C) \psi_i') dx$$

### Variable coefficient; linear system (full derivation)

$v = \psi_i$  and  $u = B + \sum_j c_j \psi_j$ :

$$(\alpha B' + \alpha \sum_{j \in \mathcal{I}_s} c_j \psi_j', \psi_i') = (f, \psi_i), \quad i \in \mathcal{I}_s$$

Reorder to form linear system:

$$\sum_{j \in \mathcal{I}_s} (\alpha \psi_j', \psi_i') c_j = (f, \psi_i) + (\alpha L^{-1}(D - C), \psi_i'), \quad i \in \mathcal{I}_s$$

This is  $\sum_j A_{i,j} c_j = b_i$  with

$$A_{i,j} = (\alpha \psi_j', \psi_i') = \int_{\Omega} \alpha(x) \psi_j'(x) \psi_i'(x) dx$$
$$b_i = (f, \psi_i) + (\alpha L^{-1}(D - C), \psi_i') = \int_{\Omega} \left( f \psi_i + \alpha \frac{D - C}{L} \psi_i' \right) dx$$

### First-order derivative in the equation and boundary condition; problem

$$-u''(x) + bu'(x) = f(x), \quad x \in \Omega = [0, L], \quad u(0) = C, \quad u'(L) = E$$

New features:

- first-order derivative  $u'$  in the equation
- boundary condition with  $u'$ :  $u'(L) = E$

Initial steps:

- Must force  $\psi_i(0) = 0$  because of Dirichlet condition at  $x = 0$
- Boundary function:  $B(x) = C(L - x)$  or just  $B(x) = C$
- No requirements on  $\psi_i(L)$  (no Dirichlet condition at  $x = L$ )

### First-order derivative in the equation and boundary condition; details

$$u = C + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x)$$

Galerkin's method: multiply by  $v$ , integrate over  $\Omega$ , integrate by parts.

$$(-u'' + bu' - f, v) = 0, \quad \forall v \in V$$

$$(u', v') + (bu', v) = (f, v) + [u'v]_0^L, \quad \forall v \in V$$

$$[u'v]_0^L = u'(L)v(L) - u'(0)v(0) = Ev(L) \text{ since } v(0) = 0 \text{ and } u'(L) = E$$

$$(u', v') + (bu', v) = (f, v) + Ev(L), \quad \forall v \in V$$

### First-order derivative in the equation and boundary condition; observations

$$(u'v') + (bu', v) = (f, v) + Ev(L), \quad \forall v \in V$$

Important observations:

- The boundary term can be used to implement Neumann conditions
- Forgetting the boundary term implies the condition  $u' = 0$  (!)
- Such conditions are called *natural boundary conditions*

## First-order derivative in the equation and boundary condition; abstract notation (optional)

Abstract notation:

$$a(u, v) = L(v) \quad \forall v \in V$$

With

$$(u'v') + (bu', v) = (f, v) + Ev(L), \quad \forall v \in V$$

we have

$$\begin{aligned} a(u, v) &= (u', v') + (bu', v) \\ L(v) &= (f, v) + Ev(L) \end{aligned}$$

## First-order derivative in the equation and boundary condition; linear system

Insert  $u = C + \sum_j c_j \psi_j$  and  $v = \psi_i$  in

$$(u'v') + (bu', v) = (f, v) + Ev(L), \quad \forall v \in V$$

and manipulate to get

$$\sum_{j \in \mathcal{I}_s} \underbrace{((\psi'_j, \psi'_i) + (b\psi'_j, \psi_i))}_{A_{i,j}} c_j = \underbrace{(f, \psi_i) + E\psi_i(L)}_{b_i}, \quad i \in \mathcal{I}_s$$

Observation:  $A_{i,j}$  is not symmetric because of the term

$$(b\psi'_j, \psi_i) = \int_{\Omega} b\psi'_j \psi_i dx \neq \int_{\Omega} b\psi'_i \psi_j dx = (\psi'_i, b\psi_j)$$

## Terminology: natural and essential boundary conditions

$$(u', v') + (bu', v) = (f, v) + u'(L)v(L) - u'(0)v(0)$$

- Note: forgetting the boundary terms implies  $u'(L) = u'(0) = 0$  (unless prescribe a Dirichlet condition)
- Conditions on  $u'$  are simply inserted in the variational form and called *natural conditions*
- Conditions on  $u$  at  $x = 0$  requires modifying  $V$  (through  $\psi_i(0) = 0$ ) and are known as *essential conditions*

**Lesson learned.** It is easy to forget the boundary term when integrating by parts. That mistake may prescribe a condition on  $u'$ !

## Nonlinear coefficient; problem

Problem:

$$-(\alpha(u)u')' = f(u), \quad x \in [0, L], \quad u(0) = 0, \quad u'(L) = E$$

- $V$ : basis  $\{\psi_i\}_{i \in \mathcal{I}_s}$  with  $\psi_i(0) = 0$  because of  $u(0) = 0$
- How does the nonlinear coefficients  $\alpha(u)$  and  $f(u)$  impact the variational formulation?  
(Not much!)

## Nonlinear coefficient; variational formulation

Galerkin: multiply by  $v$ , integrate, integrate by parts

$$\int_0^L \alpha(u) \frac{du}{dx} \frac{dv}{dx} dx = \int_0^L f(u)v dx + [\alpha(u)vu']_0^L \quad \forall v \in V$$

- $\alpha(u(0))v(0)u'(0) = 0$  since  $v(0) = 0$
- $\alpha(u(L))v(L)u'(L) = \alpha(u(L))v(L)E$  since  $u'(L) = E$

$$\int_0^L \alpha(u) \frac{du}{dx} \frac{dv}{dx} v dx = \int_0^L f(u)v dx + \alpha(u(L))v(L)E \quad \forall v \in V$$

or

$$(\alpha(u)u', v') = (f(u), v) + \alpha(u(L))v(L)E \quad \forall v \in V$$

## Nonlinear coefficient; where does the nonlinearity cause challenges?

- Abstract notation: no  $a(u, v)$  and  $L(v)$  because  $a$  and  $L$  get nonlinear
- Abstract notation for nonlinear problems:  $F(u; v) = 0 \quad \forall v \in V$
- What about forming a linear system? We get a *nonlinear* system of algebraic equations
- Must use methods like Picard iteration or Newton's method to solve nonlinear algebraic equations
- But: the variational formulation was not much affected by nonlinearities



## Examples on detailed computations by hand

### Dirichlet and Neumann conditions; problem

$$-u''(x) = f(x), \quad x \in \Omega = [0, 1], \quad u'(0) = C, \quad u(1) = D$$

- Use a *global* polynomial basis  $\psi_i \sim x^i$  on  $[0, 1]$
- Because of  $u(1) = D$ :  $\psi_i(1) = 0$
- Basis:  $\psi_i(x) = (1 - x)^{i+1}$ ,  $i \in \mathcal{I}_s$
- Boundary function:  $B(x) = Dx$
- $u(x) = B(x) + \sum_{j \in \mathcal{I}_s} c_j \varphi_j = Dx + \sum_{j \in \mathcal{I}_s} c_j (1 - x)^{j+1}$

Variational formulation: find  $(u - B) \in V$  such that

$$(u, \psi'_i) = (f, \psi_i) - (B', \psi_i) - C\psi_i(0), \quad i \in \mathcal{I}_s$$

### Dirichlet and Neumann conditions; linear system

Insert  $u(x) = B(x) + \sum_{j \in \mathcal{I}_s} c_j \varphi_j$  and derive

$$\sum_{j \in \mathcal{I}_s} A_{i,j} c_j = b_i, \quad i \in \mathcal{I}_s$$

with

$$\begin{aligned} A_{i,j} &= (\psi'_j, \psi'_i) \\ b_i &= (f, \psi_i) - (D, \psi'_i) - C\psi_i(0) \end{aligned}$$

### Dirichlet and Neumann conditions; integration

$$A_{i,j} = (\psi'_j, \psi'_i) = \int_0^1 \psi'_i(x) \psi'_j(x) dx = \int_0^1 (i+1)(j+1)(1-x)^{i+j} dx$$

Choose  $f(x) = 2$ :

$$\begin{aligned} b_i &= (2, \psi_i) - (D, \psi'_i) - C\psi_i(0) \\ &= \int_0^1 (2(1-x)^{i+1} - D(i+1)(1-x)^i) dx - C\psi_i(0) \end{aligned}$$

## Dirichlet and Neumann conditions; $2 \times 2$ system

Can easily do the integrals with `sympy`.  $N = 1$  and  $\mathcal{I}_s = \{0, 1\}$ :

$$\begin{pmatrix} 1 & 1 \\ 1 & 4/3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} -C + D + 1 \\ 2/3 - C + D \end{pmatrix}$$

$$c_0 = -C + D + 2, \quad c_1 = -1,$$

$$u(x) = 1 - x^2 + D + C(x - 1) \quad (\text{exact solution})$$

## When is the numerical method is exact?

Assume that apart from boundary conditions,  $u_e$  lies in the same space  $V$  as where we seek  $u$ :

$$\begin{aligned} u &= B + \textcolor{red}{F}, \quad F \in V \\ a(B + F, v) &= L(v), \quad \forall v \in V \\ u_e &= B + \textcolor{red}{E}, \quad E \in V \\ a(B + E, v) &= L(v), \quad \forall v \in V \end{aligned}$$

$$\text{Subtract: } a(F - E, v) = 0 \Rightarrow E = F \text{ and } u = u_e$$

## Computing with finite elements

Tasks:

- Address the model problem  $-u''(x) = 2$ ,  $u(0) = u(L) = 0$
- Uniform finite element mesh with P1 elements
- Show all finite element computations in detail

## Variational formulation

$$-u''(x) = 2, \quad x \in (0, L), \quad u(0) = u(L) = 0,$$

Variational formulation:

$$(u', v') = (2, v) \quad \forall v \in V$$

## How to deal with the boundary conditions?

Since  $u(0) = 0$  and  $u(L) = 0$ , we must force

$$v(0) = v(L) = 0, \quad \psi_i(0) = \psi_i(L) = 0$$

Let's choose the obvious finite element basis:  $\psi_i = \varphi_i$ ,  $i = 0, \dots, N_n - 1$

Problem:  $\varphi_0(0) \neq 0$  and  $\varphi_{N_n-1}(L) \neq 0$

Solution: we just exclude  $\varphi_0$  and  $\varphi_{N_n-1}$  from the basis and work with

$$\psi_i = \varphi_{i+1}, \quad i = 0, \dots, N = N_n - 3$$

Introduce index mapping  $\nu(i)$ :  $\psi_i = \varphi_{\nu(i)}$

$$u = \sum_{j \in \mathcal{I}_s} c_j \varphi_{\nu(j)}, \quad i = 0, \dots, N, \quad \nu(j) = j + 1$$

Irregular numbering: more complicated  $\nu(j)$  table

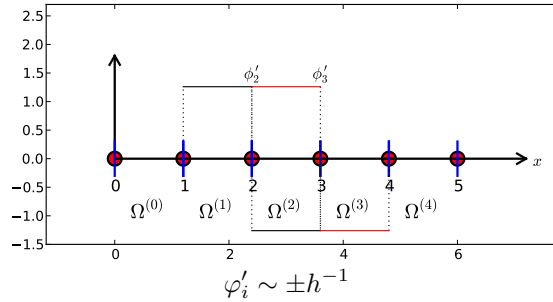
## Computation in the global physical domain; formulas

$$A_{i,j} = \int_0^L \varphi'_{i+1}(x) \varphi'_{j+1}(x) dx, \quad b_i = \int_0^L 2\varphi_{i+1}(x) dx$$

Many will prefer to change indices to obtain a  $\varphi'_i \varphi'_j$  product:  $i + 1 \rightarrow i$ ,  $j + 1 \rightarrow j$

$$A_{i-1,j-1} = \int_0^L \varphi'_i(x) \varphi'_j(x) dx, \quad b_{i-1} = \int_0^L 2\varphi_i(x) dx$$

## Computation in the global physical domain; details



$$A_{i-1,i-1} = h^{-2}2h = 2h^{-1}, \quad A_{i-1,i-2} = h^{-1}(-h^{-1})h = -h^{-1}$$

and  $A_{i-1,i} = A_{i-1,i-2}$

$$b_{i-1} = 2\left(\frac{1}{2}h + \frac{1}{2}h\right) = 2h$$

### Computation in the global physical domain; linear system

$$\frac{1}{h} \begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \end{pmatrix}$$

### Write out the corresponding difference equation

General equation at node  $i$ :

$$-\frac{1}{h}c_{i-1} + \frac{2}{h}c_i - \frac{1}{h}c_{i+1} = 2h$$

Now,  $c_i = u(x_{i+1}) \equiv u_{i+1}$ . Writing out the equation at node  $i-1$ ,

$$-\frac{1}{h}c_{i-2} + \frac{2}{h}c_{i-1} - \frac{1}{h}c_i = 2h$$

translates directly to

$$-\frac{1}{h}u_{i-1} + \frac{2}{h}u_i - \frac{1}{h}u_{i+1} = 2h$$

### Comparison with a finite difference discretization

The standard finite difference method for  $-u'' = 2$  is

$$-\frac{1}{h^2}u_{i-1} + \frac{2}{h^2}u_i - \frac{1}{h^2}u_{i+1} = 2$$

Multiply by  $h$ !

The finite element method and the finite difference method are identical *in this example*.

(Remains to study the equations at the end points, which involve boundary values - but these are also the same for the two methods)

### Cellwise computations; formulas

- Repeat the previous example, but apply the cellwise algorithm
- Work with one cell at a time
- Transform physical cell to reference cell  $X \in [-1, 1]$

$$A_{i-1,j-1}^{(e)} = \int_{\Omega^{(e)}} \varphi'_i(x) \varphi'_j(x) dx = \int_{-1}^1 \frac{d}{dx} \tilde{\varphi}_r(X) \frac{d}{dx} \tilde{\varphi}_s(X) \frac{h}{2} dX,$$

$$\tilde{\varphi}_0(X) = \frac{1}{2}(1 - X), \quad \tilde{\varphi}_1(X) = \frac{1}{2}(1 + X)$$

$$\frac{d\tilde{\varphi}_0}{dX} = -\frac{1}{2}, \quad \frac{d\tilde{\varphi}_1}{dX} = \frac{1}{2}$$

From the chain rule

$$\frac{d\tilde{\varphi}_r}{dx} = \frac{d\tilde{\varphi}_r}{dX} \frac{dX}{dx} = \frac{2}{h} \frac{d\tilde{\varphi}_r}{dX}$$

### Cellwise computations; details

$$A_{i-1,j-1}^{(e)} = \int_{\Omega^{(e)}} \varphi'_i(x) \varphi'_j(x) dx = \int_{-1}^1 \frac{2}{h} \frac{d\tilde{\varphi}_r}{dX} \frac{2}{h} \frac{d\tilde{\varphi}_s}{dX} \frac{h}{2} dX = \tilde{A}_{r,s}^{(e)}$$

$$b_{i-1}^{(e)} = \int_{\Omega^{(e)}} 2\varphi_i(x) dx = \int_{-1}^1 2\tilde{\varphi}_r(X) \frac{h}{2} dX = \tilde{b}_r^{(e)}, \quad i = q(e, r), \quad r = 0, 1$$

Must run through all  $r, s = 0, 1$  and  $r = 0, 1$  and compute each entry in the element matrix and vector:

$$\tilde{A}^{(e)} = \frac{1}{h} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \tilde{b}^{(e)} = h \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Example:

$$\tilde{A}_{0,1}^{(e)} = \int_{-1}^1 \frac{2}{h} \frac{d\tilde{\varphi}_0}{dX} \frac{2}{h} \frac{d\tilde{\varphi}_1}{dX} \frac{h}{2} dX = \frac{2}{h} \left(-\frac{1}{2}\right) \frac{2}{h} \frac{1}{2} \frac{h}{2} \int_{-1}^1 dX = -\frac{1}{h}$$

### Cellwise computations; details of boundary cells

- The boundary cells involve only one unknown
- $\Omega^{(0)}$ : left node value known, only a contribution from right node
- $\Omega^{(N_e)}$ : right node value known, only a contribution from left node

For  $e = 0$  and  $e = N_e$ :

$$\tilde{A}^{(e)} = \frac{1}{h} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \tilde{b}^{(e)} = h \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Only one degree of freedom ("node") in these cells ( $r = 0$  counts the only dof)

## Cellwise computations; assembly

4 P1 elements:

```
vertices = [0, 0.5, 1, 1.5, 2]
cells = [[0, 1], [1, 2], [2, 3], [3, 4]]
dof_map = [[0], [0, 1], [1, 2], [2]]          # only 1 dof in elm
          0, 3
```

Python code for the assembly algorithm:

```
# Ae[e][r,s]: element matrix, be[e][r]: element vector
# A[i,j]: coefficient matrix, b[i]: right-hand side

for e in range(len(Ae)):
    for r in range(Ae[e].shape[0]):
        for s in range(Ae[e].shape[1]):
            A[dof_map[e,r],dof_map[e,s]] += Ae[e][i,j]
            b[dof_map[e,r]] += be[e][i,j]
```

Result: same linear system as arose from computations in the physical domain

## General construction of a boundary function

- Now we address *nonzero Dirichlet conditions*
- $B(x)$  is not always easy to construct (i.e., extend to the interior of  $\Omega$ ), especially not in 2D and 3D
- With finite element basis functions,  $\varphi_i$ ,  $B(x)$  can be constructed in a completely general way (!)

Define

- $I_b$ : set of indices with nodes where  $u$  is known
- $U_i$ : Dirichlet value of  $u$  at node  $i$ ,  $i \in I_b$

The general formula for  $B$  is now

$$B(x) = \sum_{j \in I_b} U_j \varphi_j(x)$$

## Explanation

Suppose we have a Dirichlet condition  $u(x_k) = U_k$ ,  $k \in I_b$ :

$$u(x_k) = \sum_{j \in I_b} U_j \underbrace{\varphi_j(x)}_{\neq 0 \text{ only for } j=k} + \sum_{j \in I_s} c_j \underbrace{\varphi_{\nu(j)}(x_k)}_{=0, k \notin I_s} = U_k$$

**Example with two *nonzero* Dirichlet values; variational formulation**

$$-u'' = 2, \quad u(0) = C, \quad u(L) = D$$

$$\int_0^L u' v' \, dx = \int_0^L 2v \, dx \quad \forall v \in V$$

$$(u', v') = (2, v) \quad \forall v \in V$$

**Example with two Dirichlet values; boundary function**

$$B(x) = \sum_{j \in I_b} U_j \varphi_j(x)$$

Here  $I_b = \{0, N_n - 1\}$ ,  $U_0 = C$ ,  $U_{N_n-1} = D$ ;  $\psi_i$  are the internal  $\varphi_i$  functions:

$$\psi_i = \varphi_{\nu(i)}, \quad \nu(i) = i + 1, \quad i \in \mathcal{I}_s = \{0, \dots, N = N_n - 3\}$$

$$\begin{aligned} u(x) &= \underbrace{C \cdot \varphi_0 + D \varphi_{N_n-1}}_{B(x)} + \sum_{j \in \mathcal{I}_s} c_j \varphi_{j+1} \\ &= C \cdot \varphi_0 + D \varphi_{N_n-1} + c_0 \varphi_1 + c_1 \varphi_2 + \dots + c_N \varphi_{N_n-2} \end{aligned}$$

**Example with two Dirichlet values; details**

Insert  $u = B + \sum_j c_j \psi_j$  in variational formulation:

$$(u', v') = (2, v) \quad \Rightarrow \quad \left( \sum_j c_j \psi_j', \psi_i' \right) = (2 - B', \psi_i) \quad \forall v \in V$$

$$A_{i-1, j-1} = \int_0^L \varphi_i'(x) \varphi_j'(x) \, dx$$

$$b_{i-1} = \int_0^L (f(x) \varphi_i(x) - B'(x) \varphi_i'(x)) \, dx, \quad B'(x) = C \varphi_0'(x) + D \varphi_{N_n-1}'(x)$$

for  $i, j = 1, \dots, N + 1 = N_n - 1$ .

New boundary terms from  $-\int B' \varphi_i'$  dx:  $C/2$  for  $i = 1$  and  $-D/2$  for  $i = N_n - 2$

### Example with two Dirichlet values; cellwise computations

- All element matrices are as in the previous example
- New element vector in the first and last cell

From the last cell:

$$\tilde{b}_0^{(N_e)} = \int_{-1}^1 \left( f\tilde{\varphi}_0 - D \frac{2}{h} \frac{d\tilde{\varphi}_1}{dX} \frac{2}{h} \frac{d\tilde{\varphi}_0}{dX} \right) \frac{h}{2} dX = h + \frac{D}{h}$$

From the first cell:

$$\tilde{b}_0^{(0)} = \int_{-1}^1 \left( f\tilde{\varphi}_1 - C \frac{2}{h} \frac{d\tilde{\varphi}_0}{dX} \frac{2}{h} \frac{d\tilde{\varphi}_1}{dX} \right) \frac{h}{2} dX = h + \frac{C}{h}$$

**hpl 1:** These calculations had some errors - redo them in detail!

### Modification of the linear system; ideas

- Method 1: incorporate Dirichlet values through a  $B(x)$  function and demand  $\psi_i = 0$  where Dirichlet values apply
- Method 2: drop  $B(x)$ , drop demands to  $\psi_i$ , just assemble as if there were no Dirichlet conditions, and modify the linear system instead

Method 2: always choose  $\psi_i = \varphi_i$  for all  $i \in \mathcal{I}_s$  and set

$$u(x) = \sum_{j \in \mathcal{I}_s} c_j \varphi_j(x), \quad \mathcal{I}_s = \{0, \dots, N = N_n - 1\}$$

**Attractive way of incorporating Dirichlet conditions.**  $u$  is treated as unknown at all boundaries when computing entries in the linear system

### Modification of the linear system; original system

$$-u'' = 2, \quad u(0) = 0, \quad u(L) = D$$

Assemble as if there were no Dirichlet conditions:



$$\frac{1}{h} \begin{pmatrix} 1 & -1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & & & & \vdots \\ \vdots & & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & & 0 & -1 & 2 & -1 & \ddots \\ \vdots & & & & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} h \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \\ h \end{pmatrix}$$

### Modification of the linear system; row replacement

- Dirichlet condition  $u(x_k) = U_k$  means  $c_k = U_k$   
(since  $c_k = u(x_k)$ )
- Replace first row by  $c_0 = 0$
- Replace last row by  $c_N = D$

$$\frac{1}{h} \begin{pmatrix} h & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & & & & \vdots \\ \vdots & & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & & 0 & -1 & 2 & -1 & \ddots \\ \vdots & & & & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & & \ddots & \ddots & -1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & h \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} 0 \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h \\ D \end{pmatrix}$$

### Modification of the linear system; element matrix/vector

In cell 0 we know  $u$  for local node (degree of freedom)  $r = 0$ . Replace the first cell equation by  $\tilde{c}_0 = 0$ :

$$\tilde{A}^{(0)} = A = \frac{1}{h} \begin{pmatrix} h & 0 \\ -1 & 1 \end{pmatrix}, \quad \tilde{b}^{(0)} = \begin{pmatrix} 0 \\ h \end{pmatrix}$$

In cell  $N_e$  we know  $u$  for local node  $r = 1$ . Replace the last equation in the cell system by  $\tilde{c}_1 = D$ :

$$\tilde{A}^{(N_e)} = A = \frac{1}{h} \begin{pmatrix} 1 & -1 \\ 0 & h \end{pmatrix}, \quad \tilde{b}^{(N_e)} = \begin{pmatrix} h \\ D \end{pmatrix}$$

### Symmetric modification of the linear system; algorithm

- The modification above destroys symmetry of the matrix: e.g.,  $A_{0,1} \neq A_{1,0}$
- Symmetry is often important in 2D and 3D (faster computations, less storage)
- A more complex modification can preserve symmetry!

Algorithm for incorporating  $c_i = U_i$  in a symmetric way:

1. Subtract column  $i$  times  $U_i$  from the right-hand side
2. Zero out column and row no  $i$
3. Place 1 on the diagonal
4. Set  $b_i = U_i$

### Symmetric modification of the linear system; example

$$\frac{1}{h} \begin{pmatrix} h & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 2 & -1 & \ddots & & & & & \vdots \\ 0 & -1 & 2 & -1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & & 0 & -1 & 2 & -1 & \ddots \\ \vdots & & & & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & h \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} 0 \\ 2h \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 2h + \frac{D}{h} \\ D \end{pmatrix}$$

### Symmetric modification of the linear system; element level

Symmetric modification applied to  $\tilde{A}^{(N_e)}$ :

$$\tilde{A}^{(N_e)} = A = \frac{1}{h} \begin{pmatrix} 1 & 0 \\ 0 & h \end{pmatrix}, \quad \tilde{b}^{(N_e)} = \begin{pmatrix} h + D/h \\ D \end{pmatrix}$$

## Boundary conditions: specified derivative

**Neumann conditions.** How can we incorporate  $u'(0) = C$  with finite elements?

$$-u'' = f, \quad u'(0) = C, \quad u(L) = D$$

- $\psi_i(L) = 0$  because of Dirichlet condition  $u(L) = D$   
(or no demand and modify linear system)
- No demand to  $\psi_i(0)$
- The condition  $u'(0) = C$  will be handled (as usual) through a boundary term arising from integration by parts

## The variational formulation

Galerkin's method:

$$\int_0^L (u''(x) + f(x))\psi_i(x)dx = 0, \quad i \in \mathcal{I}_s$$

Integration of  $u''\psi_i$  by parts:

$$\int_0^L u'(x)\psi_i'(x)dx - (u'(L)\psi_i(L) - u'(0)\psi_i(0)) - \int_0^L f(x)\psi_i(x)dx = 0$$

- $u'(L)\psi_i(L) = 0$  since  $\psi_i(L) = 0$
- $u'(0)\psi_i(0) = C\psi_i(0)$  since  $u'(0) = C$

## Method 1: Boundary function and exclusion of Dirichlet degrees of freedom

- $\psi_i = \varphi_i, i \in \mathcal{I}_s = \{0, \dots, N = N_n - 2\}$
- $B(x) = D\varphi_{N_n-1}(x), u = B + \sum_{j=0}^N c_j\varphi_j$

$$\int_0^L u'(x)\varphi_i'(x)dx = \int_0^L f(x)\varphi_i(x)dx - C\varphi_i(0), \quad i \in \mathcal{I}_s$$

$$\sum_{j=0}^N \left( \int_0^L \varphi_i'\varphi_j'dx \right) c_j = \int_0^L (f\varphi_i - D\varphi_N'\varphi_i)dx - C\varphi_i(0)$$

for  $i = 0, \dots, N = N_n - 2$ .

## Method 2: Use all $\varphi_i$ and insert the Dirichlet condition in the linear system

- Now  $\psi_i = \varphi_i$ ,  $i = 0, \dots, N = N_n - 1$  (all nodes)
- $\varphi_N(L) \neq 0$ , so  $u'(L)\varphi_N(L) \neq 0$
- However, the term  $u'(L)\varphi_N(L)$  in  $b_N$  *will be erased* when we insert the Dirichlet value in  $b_N = D$

We can therefore forget about the term  $u'(L)\varphi_i(L)$ !

Boundary terms  $u'\varphi_i$  at points  $x_i$  where Dirichlet values apply can always be forgotten.

$$u(x) = \sum_{j=0}^{N=N_n-1} c_j \varphi_j(x)$$

$$\sum_{j=0}^{N=N_n-1} \left( \int_0^L \varphi'_i(x) \varphi'_j(x) dx \right) c_j = \int_0^L f(x) \varphi_i(x) dx - C \varphi_i(0)$$

Assemble entries for  $i = 0, \dots, N = N_n - 1$  and then modify the last equation to  $c_N = D$

## How the Neumann condition impacts the element matrix and vector

The extra term  $C\varphi_0(0)$  affects only the element vector from the first cell since  $\varphi_0 = 0$  on all other cells.

$$\tilde{A}^{(0)} = A = \frac{1}{h} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \quad \tilde{b}^{(0)} = \begin{pmatrix} h - C \\ h \end{pmatrix}$$

## The finite element algorithm

The differential equation problem defines the integrals in the variational formulation.

Request these functions from the user:

```
integrand_lhs(phi, r, s, x)
boundary_lhs(phi, r, s, x)
integrand_rhs(phi, r, x)
boundary_rhs(phi, r, x)
```

Must also have a mesh with `vertices`, `cells`, and `dof_map`

## Python pseudo code; the element matrix and vector

```
<Declare global matrix, global rhs: A, b>

# Loop over all cells
for e in range(len(cells)):

    # Compute element matrix and vector
    n = len(dof_map[e]) # no of dofs in this element
    h = vertices[cells[e][1]] - vertices[cells[e][0]]
    <Declare element matrix, element vector: A_e, b_e>

    # Integrate over the reference cell
    points, weights = <numerical integration rule>
    for X, w in zip(points, weights):
        phi = <basis functions + derivatives at X>
        detJ = h/2
        x = <affine mapping from X>
        for r in range(n):
            for s in range(n):
                A_e[r,s] += integrand_lhs(phi, r, s, x)*detJ*w
                b_e[r] += integrand_rhs(phi, r, x)*detJ*w

    # Add boundary terms
    for r in range(n):
        for s in range(n):
            A_e[r,s] += boundary_lhs(phi, r, s, x)*detJ*w
            b_e[r] += boundary_rhs(phi, r, x)*detJ*w
```

## Python pseudo code; boundary conditions and assembly

```
for e in range(len(cells)):
    ...

    # Incorporate essential boundary conditions
    for r in range(n):
        global_dof = dof_map[e][r]
        if global_dof in essbc_dofs:
            # dof r is subject to an essential condition
            value = essbc_docs[global_dof]
            # Symmetric modification
            b_e -= value*A_e[:,r]
            A_e[r,:] = 0
            A_e[:,r] = 0
            A_e[r,r] = 1
            b_e[r] = value

    # Assemble
    for r in range(n):
        for s in range(n):
            A[dof_map[e][r], dof_map[e][r]] += A_e[r,s]
            b[dof_map[e][r]] += b_e[r]

<solve linear system>
```

## Variational formulations in 2D and 3D

How to do integration by parts is the major difference when moving to 2D and 3D.

### Integration by parts

**Rule for multi-dimensional integration by parts.**

$$-\int_{\Omega} \nabla \cdot (a(\mathbf{x}) \nabla u) v \, d\mathbf{x} = \int_{\Omega} a(\mathbf{x}) \nabla u \cdot \nabla v \, d\mathbf{x} - \int_{\partial\Omega} a \frac{\partial u}{\partial n} v \, ds$$

- $\int_{\Omega}() \, d\mathbf{x}$ : area (2D) or volume (3D) integral
- $\int_{\partial\Omega}() \, ds$ : line(2D) or surface (3D) integral
- $\partial\Omega_N$ : Neumann conditions  $-a \frac{\partial u}{\partial n} = g$
- $\partial\Omega_D$ : Dirichlet conditions  $u = u_0$
- $v \in V$  must vanish on  $\partial\Omega_D$  (in method 1)

### Example on integration by parts; problem

$$\begin{aligned} \mathbf{v} \cdot \nabla u + \alpha u &= \nabla \cdot (a \nabla u) + f, & \mathbf{x} \in \Omega \\ u &= u_0, & \mathbf{x} \in \partial\Omega_D \\ -a \frac{\partial u}{\partial n} &= g, & \mathbf{x} \in \partial\Omega_N \end{aligned}$$

- Known:  $a$ ,  $\alpha$ ,  $f$ ,  $u_0$ , and  $g$ .
- Second-order PDE: must have *exactly one boundary condition at each point of the boundary*

Method 1 with boundary function and  $\psi_i = 0$  on  $\partial\Omega_D$ :

$$u(\mathbf{x}) = B(\mathbf{x}) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(\mathbf{x}), \quad B(\mathbf{x}) = u_0(\mathbf{x})$$

### Example on integration by parts in 1D/2D/3D

Galerkin's method: multiply by  $v \in V$  and integrate over  $\Omega$ ,

$$\int_{\Omega} (\mathbf{v} \cdot \nabla u + \alpha u) v \, d\mathbf{x} = \int_{\Omega} \nabla \cdot (a \nabla u) v \, d\mathbf{x} + \int_{\Omega} f v \, d\mathbf{x}$$

Integrate the second-order term by parts:

$$\int_{\Omega} \nabla \cdot (a \nabla u) v \, dx = - \int_{\Omega} a \nabla u \cdot \nabla v \, dx + \int_{\partial\Omega} a \frac{\partial u}{\partial n} v \, ds,$$

Result:

$$\int_{\Omega} (\mathbf{v} \cdot \nabla u + \alpha u) v \, dx = - \int_{\Omega} a \nabla u \cdot \nabla v \, dx + \int_{\partial\Omega} a \frac{\partial u}{\partial n} v \, ds + \int_{\Omega} f v \, dx$$

### Incorporation of the Neumann condition in the variational formulation

Note:  $v \neq 0$  only on  $\partial\Omega_N$  (since  $v = 0$  on  $\partial\Omega_D$ ):

$$\int_{\partial\Omega} a \frac{\partial u}{\partial n} v \, ds = \int_{\partial\Omega_N} \underbrace{a \frac{\partial u}{\partial n}}_{-g} v \, ds = - \int_{\partial\Omega_N} g v \, ds$$

The final variational form:

$$\int_{\Omega} (\mathbf{v} \cdot \nabla u + \alpha u) v \, dx = - \int_{\Omega} a \nabla u \cdot \nabla v \, dx - \int_{\partial\Omega_N} g v \, ds + \int_{\Omega} f v \, dx$$

Or with inner product notation:

$$(\mathbf{v} \cdot \nabla u, v) + (\alpha u, v) = -(a \nabla u, \nabla v) - (g, v)_N + (f, v)$$

$(g, v)_N$ : line or surface integral over  $\partial\Omega_N$ .

### Derivation of the linear system

- $\forall v \in V$  is replaced by for all  $\psi_i, i \in \mathcal{I}_s$
- Insert  $u = B + \sum_{j \in \mathcal{I}_s} c_j \psi_j, B = u_0$ , in the variational form
- Identify  $i, j$  terms (matrix) and  $i$  terms (right-hand side)
- Write on form  $\sum_{i \in \mathcal{I}_s} A_{i,j} c_j = b_i, i \in \mathcal{I}_s$

$$A_{i,j} = (\mathbf{v} \cdot \nabla \psi_j, \psi_i) + (\alpha \psi_j, \psi_i) + (a \nabla \psi_j, \nabla \psi_i)$$

$$b_i = (g, \psi_i)_N + (f, \psi_i) - (\mathbf{v} \cdot \nabla u_0, \psi_i) + (\alpha u_0, \psi_i) + (a \nabla u_0, \nabla \psi_i)$$

### Transformation to a reference cell in 2D/3D (1)

We want to compute an integral in the physical domain by integrating over the reference cell.

$$\int_{\Omega(e)} a(\mathbf{x}) \nabla \varphi_i \cdot \nabla \varphi_j \, dx$$

Mapping from reference to physical coordinates:

$$\mathbf{x}(\mathbf{X})$$

with Jacobian  $J$ ,

$$J_{i,j} = \frac{\partial x_j}{\partial X_i}$$

### Transformation to a reference cell in 2D/3D (2)

- $dx \rightarrow \det J \, dX$ .
- Must express  $\nabla \varphi_i$  by an expression with  $\tilde{\varphi}_r$ ,  $i = q(e, r)$ :  $\nabla \tilde{\varphi}_r(\mathbf{X})$
- We want  $\nabla_{\mathbf{x}} \tilde{\varphi}_r(\mathbf{X})$  (derivatives wrt  $\mathbf{x}$ )
- What we readily have is  $\nabla_{\mathbf{X}} \tilde{\varphi}_r(\mathbf{X})$  (derivative wrt  $\mathbf{X}$ )
- Need to transform  $\nabla_{\mathbf{X}} \tilde{\varphi}_r(\mathbf{X})$  to  $\nabla_{\mathbf{x}} \tilde{\varphi}_r(\mathbf{X})$

### Transformation to a reference cell in 2D/3D (3)

Can derive

$$\begin{aligned} \nabla_{\mathbf{X}} \tilde{\varphi}_r &= J \cdot \nabla_{\mathbf{x}} \varphi_i \\ \nabla_{\mathbf{x}} \varphi_i &= \nabla_{\mathbf{x}} \tilde{\varphi}_r(\mathbf{X}) = J^{-1} \cdot \nabla_{\mathbf{X}} \tilde{\varphi}_r(\mathbf{X}) \end{aligned}$$

Integral transformation from physical to reference coordinates:

$$\int_{\Omega(e)} a(\mathbf{x}) \nabla_{\mathbf{x}} \varphi_i \cdot \nabla_{\mathbf{x}} \varphi_j \, dx = \int_{\tilde{\Omega}^r} a(\mathbf{x}(\mathbf{X})) (J^{-1} \cdot \nabla_{\mathbf{X}} \tilde{\varphi}_r) \cdot (J^{-1} \cdot \nabla_{\mathbf{X}} \tilde{\varphi}_s) \det J \, dX$$



## Numerical integration

Numerical integration over reference cell triangles and tetrahedra:

$$\int_{\tilde{\Omega}^r} g \, dX = \sum_{j=0}^{n-1} w_j g(\bar{\mathbf{X}}_j)$$

Module `numint.py` contains different rules:

```
>> import numint
>> x, w = numint.quadrature_for_triangles(num_points=3)
>> x
[(0.16666666666666666, 0.16666666666666666),
 (0.6666666666666666, 0.16666666666666666),
 (0.16666666666666666, 0.66666666666666666)]
>> w
[0.16666666666666666, 0.16666666666666666, 0.16666666666666666]
```

- Triangle: rules with  $n = 1, 3, 4, 7$  integrate exactly polynomials of degree 1, 2, 3, 4, resp.
- Tetrahedron: rules with  $n = 1, 4, 5, 11$  integrate exactly polynomials of degree 1, 2, 3, 4, resp.